Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates

Martin G. Skjæveland[1], Henrik Forssell[1], Johan W. Klüwer[2], Daniel Lupp[1], Evgenij Thorstensen[1], and Arild Waaler[1]

[1] Department of Informatics, University of Oslo
{martige,jonf,danielup,evgenit,arild}@ifi.uio.no
[2] DNV GL, Norway
johan.wilhelm.kluewer@dnvgl.com

Abstract. Reasonable Ontology Templates, OTTRs for short, are OWL ontology macros capable of representing ontology design patterns (ODPs) and closely integrating their use into ontology engineering. An OTTR is itself an OWL ontology or RDF graph, annotated with a special purpose OWL vocabulary. This allows OTTRs to be edited, debugged, published, identified, instantiated, combined, used as queries and bulk transformations, and maintained—all leveraging existing W3C standards, best practices and tools. We show how such templates can drive a technical framework and tools for a practical, efficient and transparent use of ODPs in ontology design and instantiation. The framework allows for a clear separation of the design of an ontology, typically managed by ontology experts, and its bulk content, provided by domain experts. We illustrate the approach by reconstructing the published Chess Game ODP and producing linked chess data.

1 Introduction

Ontology-based methods have matured to where they offer knowledge workers practical solutions for data management. In particular, tools that support W3C recommendations, such as reasoning tools for OWL ontologies, are sufficiently stable and efficient to allow wide-scale industrial use. However, from the perspective of product vendors and consultancy companies in the IT industry, ontologies are still viewed as a fringe technology. Hence ontology-based solutions are rarely proposed to enterprise customers, and support from the software industry is quite limited. One factor that impedes uptake is the high cost of establishing and maintaining a high-quality ontology. In part this is due to the scarcity of ontology experts, with availability in most cases below critical mass.

Ontology design patterns (ODPs) [9] serve the purpose of alleviating some of the difficulties involved with creating ontologies by offering reusable, best-practice building blocks and structures for ontology construction, commonly implemented and published as small OWL ontologies. Methods for combining and instantiating ODPs are described [20,8], and a methodology for building ontologies using patterns exists [1]. However, while ODPs are often presented as "practical building blocks" [20], we argue that ODPs in their current form, i.e., as found on http://ontologydesignpatterns.org/ featuring a graphical representation, a description and a "reusable OWL building block", are not practical enough, especially for the development of large ontologies—as using and adapting ODPs to a particular modelling task currently often require considerable manual work. What is

needed are better tool supported methods for applying ODPs in ontology engineering. Efficient tool support is imperative to industrial scale deployment of ontology-based methods.

The work reported on in this paper has the potential to remedy the situation; *Reasonable Ontology Templates* (OTTRs) are simple, but powerful, templates or macros for ontologies, cf. [21], represented in OWL using a dedicated OWL vocabulary. An OTTR can be viewed as a *parameterised ontology* which can be nested, i.e., defined using other OTTRs, and *instantiated* by providing arguments to fit the parameters of the template. By recursively *expanding* an OTTR by replacing any containing OTTR with the pattern it represents, we obtain a regular OWL ontology. Using this feature, we can reason both on the OTTR specification and its expansion, and additionally leverage existing W3C languages and tools for different ontology engineering tasks—all driven by OTTRs. Specifically, the implicit mapping between an OTTR's parameters and its pattern may be exploited to *generate* various format descriptions and transformation specifications, e.g., queries for extracting pattern instances and transformations between tabular input formats and OTTR pattern instances that may be processed by readily available desktop tools. The only additional tool support that is needed to make use of OTTRs, are tools that can perform the relatively simple operation of template expansion and instantiation.

In addition to supporting the work of the ontology engineer, we believe OTTRs can provide a framework whereby a few ontology experts can serve a large number of domain experts and put these in position to actively contribute to the development and maintenance of ontologies. This is achieved by clearly separating the design of an ontology and its bulk content: The ontology expert designs and combines patterns represented as OTTRs to provide *user-facing* patterns on a level of abstraction suitable for the domain experts. From the user-facing OTTRs a simple input format is generated together with a transformation specification of the input format to ontology format. The task of the domain expert is then "only" to provide instance arguments to the input format.

Sec. 2 defines OTTR templates and introduces the OTTR OWL vocabulary for expressing them for use on the semantic web. Furthermore, we explain how OTTRs may be used for driving a technical framework for different ontology engineering tasks, and also give a prototype implementation that can serve the various specifications for such a framework. Sec. 3 discusses the particular application of OTTRs for using ODPs for ontology design and instantiation, illustrated on the Chess Game ODP, and for linked data publication. In Sec. 4 we discuss the benefits and shortcomings for OTTRs and compare with related work. We conclude with Sec. 5.

2 Reasonable Ontology Templates

In the following we introduce the core concepts *template*, *template instance* and *expansion* through examples and by alluding to basic description logic concepts; see [6] for a more formal and thorough description.

A *template* $\mathcal{T}$ is a knowledge base $\mathcal{O}_\mathcal{T}$ and a list of parameters $\mathrm{param}(\mathcal{T}) = (\mathsf{p}_1, \ldots, \mathsf{p}_\mathsf{n})$ of distinguished concept, role, or individual names from $\mathcal{O}_\mathcal{T}$. We write a template as

$$\mathcal{T}(\mathsf{p}_1, \ldots, \mathsf{p}_\mathsf{n}) :: \mathcal{O}_\mathcal{T}.$$

and refer to the left side as the *head* and the right side as the *body*. For a list of parameters $(q_1, \ldots, q_n)$ we call $\mathcal{T}(q_1, \ldots, q_n)$ a *template instance*. Intuitively, a template instance is shorthand for representing a specific occurrence or instance of a pattern. More precisely, the *expansion* of $\mathcal{T}(q_1, \ldots, q_n)$ is the ontology $\mathcal{O}_\mathcal{T}(q_1, \ldots, q_n)$ obtained by replacing each parameter occurrence of $\mathsf{p_i}$ in $\mathcal{O}_\mathcal{T}$ with the argument q_i, for all $1 \leq i \leq n$.

Example 1. $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}) :: \{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$ is the template PartOf which has a single axiom knowledge base $\{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$ where hasPart is a role name and part and whole are parameters. $\mathsf{PartOf}(\mathrm{SteeringWheel}, \mathrm{Car})$ is an instance of PartOf representing the ontology $\{\mathrm{Car} \sqsubseteq \exists \mathrm{hasPart}.\mathrm{SteeringWheel}\}$. Note that also $\mathsf{PartOf}(\mathsf{part}, \mathsf{whole})$ is an instance of PartOf, where the parameter names are substituted for themselves; its ontology is $\{\mathsf{whole} \sqsubseteq \exists \mathrm{hasPart}.\mathsf{part}\}$.

In addition to ontology axioms, a template may also contain template instances in its body. The notion of template instance expansion is then extended to a recursive operation where any template instances in the template body are expanded tail-recursively. Cyclic template definitions are not allowed.

Example 2. Let QualityValue be the template

$$\mathsf{QualityValue}(\mathsf{x}, \mathsf{hasQuality}, \mathsf{uom}, \mathsf{val}) :: \\ \{\mathsf{x} \sqsubseteq \exists \mathsf{hasQuality}.(\forall \mathrm{hasDatum}.(\exists \mathrm{hasUOM}.\{\mathsf{uom}\} \sqcap \exists \mathrm{hasValue}.\{\mathsf{val}\}))\}$$

which intuitively expresses that x has a quality with a given value val with a given unit of measurement uom. Using this template and the PartOf template from Ex. 1, and fixing some of the parameters, the template PartLength is defined as

$$\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length}) :: \{\mathsf{PartOf}(\mathsf{part}, \mathsf{whole}), \\ \mathsf{QualityValue}(\mathsf{part}, \mathrm{hasLength}, \mathrm{meter}, \mathsf{length})\}$$

which expresses that the whole has a part with a given length measured in meters. An example instance of the template is $\mathsf{PartLength}(\mathrm{2CV}, \mathrm{SoftTop}, 1.40)$ stating that (the car) 2CV has a softtop (roof) of length 1.40 meters. The expansion of the instance $\mathsf{PartLength}(\mathsf{whole}, \mathsf{part}, \mathsf{length})$ is

$$\{\mathrm{2CV} \sqsubseteq \exists \mathrm{hasPart}.\mathrm{SoftTop}, \\ \mathrm{SoftTop} \sqsubseteq \exists \mathrm{hasLength}.(\forall \mathrm{hasDatum}.(\exists \mathrm{hasUOM}.\{\mathrm{meter}\} \sqcap \exists \mathrm{hasValue}.\{1.40\}))\}.$$

2.1 The OTTR OWL vocabulary

We adapt ontology templates to the semantic web by serialising them using RDF with the OTTR[3] OWL vocabulary defined specifically for this task. A template is associated with an RDF graph [3] (document) available at the IRI of the template.[4] The RDF graph contains both the head, identifying the template and its parameters, and the body of

[3] We use OTTR to designate the vocabulary. All other unprefixed, inline typewriter font words refer to OTTR vocabulary elements.

[4] Similarly as for OWL ontologies [17, section 3.2 Ontology Documents]

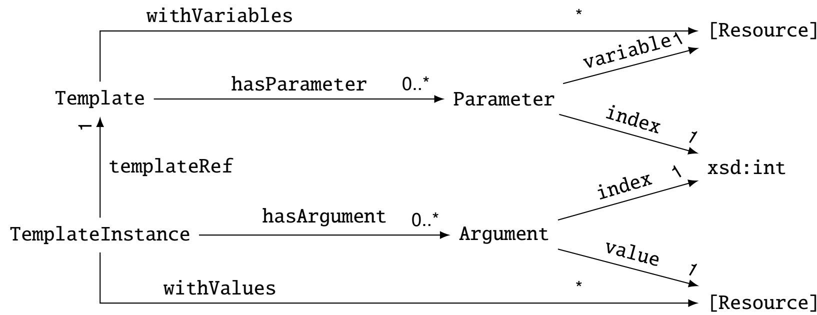

Fig. 1: High-level overview of the OTTR OWL vocabulary.

the template. The template body may contain template instances and other ontology axioms, which are expressed using regular RDF/OWL serialisation [19]. (Note that the RDF graph need not represent an OWL ontology. In fact, templates may be used in a more generic way as "RDF macros". However, we prefer to call them OWL macros in order to clearly indicate their applicability to ontology engineering, reasoning and ontology design patterns.) Parameters and arguments are represented as named variables and named values, respectively, where the name is represented in RDF as an IRI, and variables and values may be any RDF resource, i.e., any IRI or literal [3].

The most prominent elements of the OTTR vocabulary are the following, see also Fig. 1 for a graphical overview. A `Template` has zero or more `Parameter`s. Each `Parameter` is consecutively numbered by an integer valued `index`, starting at 1, and has a `variable` which represents the parameter in the template body. The existence of a `Template` in an RDF graph declares the graph as a template, and an RDF graph specifying a template must contain only one `Template` object. A `TemplateInstance` must refer to a single `Template` via a `templateReference`, and have `Argument`s to match the `Parameter`s of the `Template`. An `Argument` must have a `value` and refer a `Parameter` by using the same `index` value as the `Parameter`'s. The range of the `variable` and `value` properties is any RDF resource. These conditions and more are represented in OWL using the OTTR vocabulary available from `http://ns.ottr.xyz/`.

We differentiate between the head and the body of a template represented in RDF using the concept of graph neighbourhood, which informally are all the outgoing triples from the template and parameter individuals.

Definition 1. *Let G be an RDF graph (represented as a set of triples), and r an IRI. We define the* out-neighbourhood of r in G, *denoted* $\mathsf{out}(r, G)$ *as the set of triples* $\langle r, x, y\rangle \in G$. *Let G be the RDF graph associated with a template with IRI t and parameter IRIs $p_1, \ldots, p_n$. We define the* head *of the template in G as* $\mathsf{head}(t) = \bigcup_{x \in \{t, p_1, \ldots, p_n\}} \mathsf{out}(x, G)$, *and the* body *of t in G as* $\mathsf{body}(t) = G \setminus \mathsf{head}(t)$.

A template instance is expanded by copying the template RDF graph[5] to which the instance refers and for each template parameter substituting *all* occurrences of the parameter variable in the RDF graph with its matching argument value. The expansion is applied recursively.

[5] Also useful is the expansion procedure that only copies the template body.

Example 3. The PartOf template from Ex. 1 is represented is the OTTR vocabulary as follows:[6]

```
@prefix ottr:    <http://ns.ottr.xyz/templates#> .
@prefix partOf:  <http://www.ontologydesignpatterns.org/cp/owl/partof.owl#> .
@prefix :        <http://draft.ottr.xyz/i17/partof#> .
  ### head:
<http://draft.ottr.xyz/i17/partof> a owl:Ontology , ottr:Template ;
    owl:imports <http://www.ontologydesignpatterns.org/cp/owl/partof.owl> ;
    ottr:hasParameter [ ottr:index 1; ottr:variable :Whole ] ,
                      [ ottr:index 2; ottr:variable :Part  ] .
  ### body:
:Part  a owl:Class .
:Whole a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
        owl:onProperty partOf:hasPart ; owl:someValuesFrom :Part ] .
```

Observe that the RDF graph is a regular OWL ontology using the OTTR vocabulary to identify the template and its parameters: The template contains a head and body as indicated by the comments, and specifies two parameters with respectively the variables `:Whole` and `:Part`. These variables are used in the template body as regular RDF resources. An instance of this template, reflecting the instance in Ex. 1, is represented as follows.

```
[] ottr:templateRef <http://draft.ottr.xyz/i17/partof> ;
   ottr:hasArgument [ ottr:index 1 ; ottr:value ex:Car ] ,
                    [ ottr:index 2 ; ottr:value ex:SteeringWheel ] .
```

The instance refers to the template with `templateRef`, and each argument refers to a parameter of the template using indices. The expansion of the instance is created by replacing, in a copy of the template RDF graph, all occurrences of `:Whole` with `ex:Car`, and `:Part` with `ex:SteeringWheel`.

In order to support a more terse specification of templates and instances, the OTTR vocabulary allows for the use of RDF lists [3] to specify template parameters and instance arguments. Since the RDF list structure is reserved for the serialisation of OWL and therefore not permissible for use in valid OWL2 DL ontologies, the OTTR vocabulary also defines a linked list structure [4], called `List`. Lists may be serialised using either RDF lists or OTTR's `Lists`. The list feature may be used for directly giving the parameter variables of a template, using `withVariables`; and the argument values of an instance, using `withValues`.

Example 4. Using the RDF list format, the template PartLength given in Ex. 2 can be compactly represented:

```
<http://draft.ottr.xyz/i17/partLength> a owl:Ontology , ottr:Template ;
    ottr:withVariables ( :Whole :Part 99 )
[]  ottr:templateRef <http://draft.ottr.xyz/i17/partof> ;
    ottr:withValues ( :Whole :Part ) .
[]  ottr:templateRef <http://draft.ottr.xyz/i17/qualityvalue> ;
    ottr:withValues ( :Part ex:hasLength ex:meter 99 ).
```

[6] Note that all example templates are published at their IRI address.

Lists may also be used as argument values, supporting patterns which naturally permit variable sized input. Expanding an instance of a template allowing list input, will for each list valued argument replace all occurrences of lists in the template with the same contents as the matching parameter value list.

Example 5. The EquivObjectUnionOf template takes two arguments, a class U and a list of classes, and defines the union of the list of classes as equivalent to U.

```
<http://candidate.ottr.xyz/owl/axiom/EquivObjectUnionOf> a ottr:Template ;
    ottr:withVariables ( :U ( :A :B ) ).
:U rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Class ; owl:unionOf ( :A :B ) ] .
```

Notice that we here use the list format both to specify the template's two variables, and the second parameter's list variable. An example instance of this template is the following.

```
[]  ottr:templateRef <http://candidate.ottr.xyz/owl/axiom/EquivObjectUnionOf> ;
    ottr:withValues ( ex:Fruit ( ex:Apple ex:Orange ex:Melon ) ).
```

When expanding the instance, all occurrences of lists with the contents `:A :B` in the template will be replaced with a list (copy) containing the elements `ex:Apple ex:Orange ex:Melon`.

In addition to providing a vocabulary for expressing templates, the OTTR ontology includes axioms that allows regular ontology reasoners to check the consistency of OTTR templates, such as domain and range axioms, and functional and key constraints of properties. Also, each parameter variable may be assigned a *type* using different "variable" properties, such as `classVariable`, which informally are subproperties of the `variable` property.[7] The available types reflect the generic classes from the RDF(S) [2] and OWL [19] vocabularies and are arranged in a taxonomy accordingly, where many types are made incompatible using disjoint property ranges; consult the OTTR vocabulary for details. This simple type checking feature is very useful when constructing templates which typically pass on parameters as arguments to other templates, allowing a parameter to be assigned multiple, and possibly incompatible, types. An ontology reasoner will reveal such an inconsistency by reasoning on the OTTR vocabulary of the expanded template. An implementation of the template mechanism should also exploit the type information to check whether instance arguments respect the type of their matching parameter.

Example 6. Assume the PartOf template from Ex. 3 types both parameter variables as classes, using the `classVariable` property, here showing only the head:

```
<http://draft.ottr.xyz/i17/partof> a owl:Ontology , ottr:Template ;
   ottr:hasParameter [ ottr:index 1; ottr:classVariable :Whole ] ,
                     [ ottr:index 2; ottr:classVariable :Part  ] .
```

[7] We say "informal" since, in order to support reasoning, the specialisations of the OWL annotation property `variable` are either object properties or datatype properties, and these OWL property types are mutually disjoint.

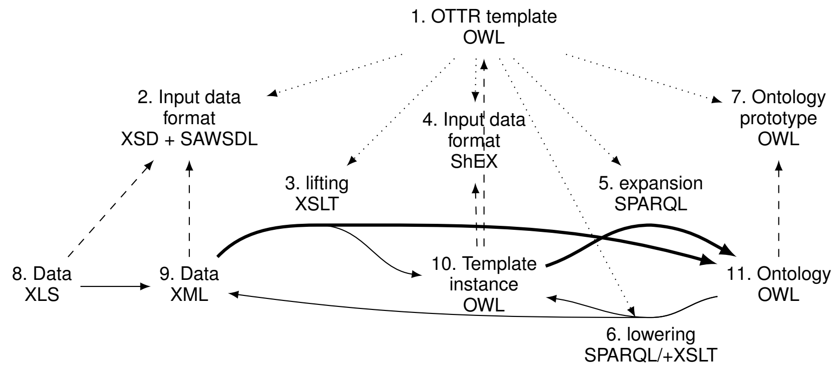

Fig. 2: OTTR-driven ontology construction. The nodes in the diagram each represent a document used in the process. All data format and transformation W3C specifications (2–7) are generated from a template specification, as indicated by the dotted edges. The dashed edges indicate an "instance of" relation, and the solid edges show the flow of the ontology bulk data, highlighting the main routes. The XSD document (2) specifies a "tabular" template instance data input format for XML data (9) that is also supported by some spreadsheet applications (8). The XML data is lifted [5] using XSLT transformations (3) to either the RDF/OWL template instance format (10) which may be validated by a ShEx shape expression (4), or directly to a regular OWL ontology (11). Instances (10) may also be expanded with generated SPARQL queries (5), and be extracted and lowered [5] with SPARQL and XSLT (6) to OWL/RDF and XML format, respectively. The lifting (3) and lowering (6) scripts are identified using SAWSDL in (2).

Now assume the variable of the first parameter of the PartLength template in Ex. 4 is (wrongly) typed as an annotation property using `annotationPropertyVariable`. Then PartLength is inconsistent, since its `:Whole` variable is classified as the disjoint classes `Class` and `AnnotationProperty`.

The OTTR ontology is defined by two ontology documents: *templates-lite* and *templates-core*. The former declares only the vocabulary with domain and range axioms and is useful when the task is primarily to instantiate templates, hence reasoning over template specifications is usually not required. The latter ontology imports the first and enables the reasoning capabilities presented above.

2.2 Ontology Construction Framework

A core feature of OTTR templates is the ability to relate a simple tabular input format, the template head, to a rich ontological structure in the template body, possibly specified via compositions of other templates. The fact that a template specifies both a tabular input format, an output ontology, and a mapping between the two formats may be exploited by leveraging the capabilities of existing W3C standards and implementations: In addition to specifying an ontology representing a prototypical ontology of the template (the expanded template) in OWL, a template can specify tabular and graph input format specifications using, e.g., XSD and ShEx, and transformations to and from (liftings and lowerings [5]) the ontology output format using, e.g., XSLT and SPARQL. This means that data can be

captured in bulk using XML- or XSD-aware client tools, and efficiently processed using XSLT and/or SPARQL processors, all of which are driven by specifications generated from a template. The process is illustrated and explained in Fig. 2.

Using this framework the ontology engineering task can be split in two more or less distinct responsibilities: one managed by the ontology engineer and the other by the domain matter expert. The main task of the ontology engineer is to design and maintain a library of interconnected templates capable of capturing the knowledge of the domain matter expert at the correct abstraction level and using a vocabulary and format recognisable by the expert. The specificity needed for the ontology engineering task at hand is achieved by iteratively combining and composing basic and more complex templates to result in *user-facing* templates. From such templates, tabular input format specification and transformations may be generated from the template specification, presenting a simple tabular format for the user to fill in which can be transformed directly to OWL ontology format using readily available desktop tools.

This process ensures uniformity and completeness of the captured domain knowledge: completeness, as the template specifies all the attributes that are necessary and variable; and uniformity as the template instances are guaranteed to expand to the desired patterns. The correctness of templates may be secured by consistency checking the prototype ontology resulting from expanding the template, as well as for each of the comprising templates in isolation. Additional syntactic constraints on the input data may be specified for the input formats which also can be used to check completeness of the input data, provided the format specification works under closed-world semantics.

2.3 Implementation

A prototype implementation that interprets the OTTR vocabulary and provides parts of the technical framework presented in Sec. 2.2 is available as an online web application and as a feature-limited standalone Java application from `http://www.ottr.xyz`.

Provided an OTTR template, specified with the IRI query parameter `?tpl`, the web implementation serves a variety of specifications and instances over HTTP: the template specification, the expansion, lifting and lowering queries as different types of SPARQL queries, and a simple XSD format and XML sample. Some services allow template instances to be created by providing argument values as IRI query parameters. With these services, the template and its different format specifications may be directly identified and used by other specifications, e.g., in OWL ontology `owl:import` statements.

Example 7. We encourage the reader to visit `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/i17/partlength` for a display of the PartLength template of Ex. 4. The service lists the template's parameters with type information and containing template instances, together with links to all the other available services of the web application.

3 Ontology Design Patterns vs. Ontology Templates

As argued in the introduction, we believe that ODPs in their current form are not practical ontology building blocks, in the sense of being actively and directly applicable in the

engineering of OWL ontologies. Rather they are conceptual building blocks that (only[8]) represent best practices of common modelling challenges—which of course is extremely useful. The practical ways of using an ODP in OWL ontology development are however limited: the natural possibilities are either to import its OWL implementation, which includes the whole pattern as-is, or by cloning (parts of) it. The included pattern may be further engineered with different techniques such as specialisation and generalisation [20]. The XDP [7] tool offers the possibility to instantiate ODPs using so called template-based and specialisation-based techniques [8]. However, in all these cases the process is largely manual and does not scale.

Reasonable Ontology Templates offer a technical framework that allows patterns, as represented by ODPs, to be applied to OWL ontology engineering in an efficient and transparent manner, avoiding unnecessary manual intervention. Comparing ODPs and OTTRs to software engineering, we believe that ODPs play the role of software engineering patterns, "representing general reusable solution to a commonly occurring problem within a given context[, but] not a finished design that can be transformed directly into source or machine code" [22]. In contrast, we think of a set of OTTRs as representing an application programming interface (API) for OWL ontology engineering, like the OpenGL API[9] does for graphics rendering, providing precise and transparent abstractions over the underlying OWL syntax that are directly applicable in the construction of OWL ontologies.

The idea of an API for OWL patterns has been implemented by representing the structural specification of OWL 2 to RDF graphs [19] as a set of OTTR templates. The purpose is to demonstrate that OWL ontologies may be represented completely by a set of OTTR instances, which, in its terse list input format, are arguably more readable than the RDF serialisation of OWL axioms. The EquivObjectUnionOf of Ex. 5 is an example of an OTTR template of an OWL axiom. Other examples are found in Fig. 3a. These templates, including other templates of different maturity, are published in an online library of OTTRs available at http://library.ottr.xyz and backed by open git repositories at http://www.gitlab.com/ottr.

3.1 Building Ontologies and Linked Datasets with OTTRs

To illustrate how OTTRs can be applied for ontology modelling and publication of linked data, we now demonstrate how OTTRs can be used to construct the Chess Game ODP [15] and to produce linked data representations of chess games, cf. [14].

The Chess Game ODP [15] is presented as a worked example for modelling with ODPs, using them to construct a chess game ontology intended to be used for describing chess games, i.e., who the players were, the end result, the list of moves, the chess opening, and where the game took place. To this end, the authors use adapted versions of the Agent Role and Event ODP, where the Event ODP extends the Agent Role ODP. They also make use of implicit OWL axiom macros for expressing *scoped domains* and *ranges*, and regular axioms like cardinality restrictions. Although the exposition and the graphical depictions of the chess game pattern are clear, its axiomatisation, and hence its

[8] In the literature ODPs are often represented as both best practice modelling patterns *and* practical building blocks.

[9] URL: https://www.opengl.org/

$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{A},\mathsf{R},\mathsf{B}) :: \{ \mathsf{A} \sqsubseteq \exists \mathsf{R}.\mathsf{B} \}$ $\mathcal{T}_{\sqsupseteq,\exists}(\mathsf{A},\mathsf{R},\mathsf{B}) :: \{ \exists \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$

$\mathcal{T}_{\sqsubseteq,\forall}(\mathsf{A},\mathsf{R},\mathsf{B}) :: \{ \mathsf{A} \sqsubseteq \forall \mathsf{R}.\mathsf{B} \}$ $\mathcal{T}_{\sqsupseteq,\forall}(\mathsf{A},\mathsf{R},\mathsf{B}) :: \{ \forall \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$

$\mathcal{T}_{\sqsubseteq,=}(\mathsf{A},\mathsf{i},\mathsf{R},\mathsf{B}) :: \{ \mathsf{A} \sqsubseteq =_{\mathsf{i}} \mathsf{R}.\mathsf{B} \}$ $\mathcal{T}_{\sqsupseteq,=}(\mathsf{A},\mathsf{i},\mathsf{R},\mathsf{B}) :: \{ =_{\mathsf{i}} \mathsf{R}.\mathsf{B} \sqsubseteq \mathsf{A} \}$

$\mathsf{DisjointClasses}(\langle \mathsf{C}_1, \mathsf{C}_2, \ldots \rangle) :: \{ \texttt{DisjointClasses}(\mathsf{C}_1, \mathsf{C}_2, \ldots) \}$

(a) Basic OWL axioms represented as OTTR templates.

$\mathsf{ScopedDomainRange}(\mathsf{R},\mathsf{A},\mathsf{B}) :: \{ \mathcal{T}_{\sqsupseteq,\exists}(\mathsf{A},\mathsf{R},\mathsf{B}), \mathcal{T}_{\sqsubseteq,\forall}(\mathsf{A},\mathsf{R},\mathsf{B}) \}$

$\mathsf{AgentRole}_5(\mathsf{AgentRole}, \mathsf{RoleProvider}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy}) :: \{$
$\mathsf{Range}(\mathsf{providesRole}, \mathsf{AgentRole}),$
$\mathsf{ScopedDomainRange}(\mathsf{performedBy}, \mathsf{AgentRole}, \mathsf{Agent}),$
$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{AgentRole}, \mathsf{providesRole}^{-}, \mathsf{RoleProvider})$
$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{AgentRole}, \mathsf{performedBy}, \mathsf{Agent}),$
$\mathsf{DisjointClasses}(\langle \mathsf{AgentRole}, \mathsf{Agent} \rangle)\}$

$\mathsf{Event}_{10}(\mathsf{Event}, \mathsf{subEventOf}, \mathsf{AgentRole}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy},$
$\mathsf{Place}, \mathsf{atPlace}, \mathsf{TemporalExtent}, \mathsf{atTime}) :: \{$
$\mathsf{AgentRole}_5(\mathsf{AgentRole}, \mathsf{Event}, \mathsf{providesRole}, \mathsf{Agent}, \mathsf{performedBy})$
$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{Event}, \mathsf{atPlace}, \mathsf{Place}),$
$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{Event}, \mathsf{atTime}, \mathsf{TempExt}),$
$\mathsf{ScopedDomainRange}(\mathsf{atPlace}, \mathsf{Event}, \mathsf{Place}),$
$\mathsf{ScopedDomainRange}(\mathsf{atTime}, \mathsf{Event}, \mathsf{Time}),$
$\mathsf{ScopedDomainRange}(\mathsf{subEventOf}, \mathsf{Event}, \mathsf{Event}),$
$\mathsf{DisjointClasses}(\langle \mathsf{Event}, \mathsf{Place}, \mathsf{TempExt}, \mathsf{AgentRole}, \mathsf{Agent} \rangle)\}$

(b) Template-based OTTR templates: ScopedDomainRange, AgentRole and Event.

$\mathsf{AgentRole}_2(\mathsf{xAgentRole}, \mathsf{xRoleProvider}) :: \{$
$\mathsf{xAgentRole} \sqsubseteq \mathrm{AgentRole},$
$\mathcal{T}_{\sqsubseteq,\exists}(\mathsf{xRoleProvider}, \mathrm{providesAgentRole}, \mathsf{xAgentRole}),$
$\mathcal{T}_{\sqsubseteq,=}(\mathsf{xAgentRole}, 1, \mathrm{providesAgentRole}^{-}, \mathsf{xRoleProvider})\}$

$\mathsf{Event}_2(\mathsf{xEvent}, \mathsf{xAgentRole}) :: \{$
$\mathsf{xEvent} \sqsubseteq \mathrm{Event},$
$\mathsf{AgentRole}_2(\mathsf{xAgentRole}, \mathsf{xEvent})\}$

(c) Specialisation-based OTTR templates for AgentRole and Event patterns.

```
[] ottr:templateRef <http://draft.ottr.xyz/chess/iChessGameReport> ;
   ottr:withValues (  "WCh 2013" "Chennai IND" "2013.11.09" "Carlsen, Magnus"
      "Anand, Viswanathan" "1/2-1/2" "2870" "2775" "A07" ( "Nf3" "d5" [...] ))) .
```

(d) OTTR template instance of the ChessGameReport template, including only two chess moves.

Fig. 3: OTTR templates used for the Chess Game pattern and for linked data publication.

OWL implementation,[10] reveals shortcomings of the presentation. These problems stem mainly from the fact that no abstraction mechanism that can encapsulate patterns and present clear interfaces for use is available: The agent role and event patterns are not clearly identified and encapsulated, and it is not clear which axioms belong to which patterns. (This is only made clear by the document formatting.) These patterns are also specialised, but it is difficult to identify exactly how, e.g., for which parts of the pattern are specialisations introduced. The scoped domain and range axioms are often used, and usually in pairs, but this is not easy to spot.

A sample of the OTTR templates used to reconstruct the Chess Game pattern is found in Fig. 3. The ScopedDomainRange template in Fig. 3b illustrates the how to basic OWL axiom templates (found in Fig. 3a) can be combined. The Event_{10} template succinctly presents that its definition relies on the $\mathsf{AgentRole}_5$ template and other templates, some which represent regular OWL axioms like existential restrictions ($\mathcal{T}_{\sqsubseteq,\exists}(\cdot,\cdot,\cdot)$). Note that all of these OTTR templates represent template-based instantiatations [8], which can be seen by the fact that all vocabulary elements are parameterised; the user can (and must) introduce the vocabulary, the only fixed vocabulary is the logical vocabulary. Fig. 3c contains two specialisation-based [8] OTTR templates, where (some) fixed non-logical vocabulary elements are specialised by template arguments. In our modelling of the chess game pattern we use template-based OTTRs to introduce the basic vocabulary of the pattern. This vocabulary is then used in specialisation-based OTTRs to provide the user with patterns which do not necessarily represent any "self-contained semantic unit", but merely represents a combination of often used axioms packaged in a template to avoid tedious repetitions. From this observation it follows that template-based OTTRs arguably better fit the idea of a publicly available API, while specialisation-based OTTR templates are naturally closer tied to specific ontology models. The complete Chess Game OTTR can be found at http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/ChessGame.ttl.

Krisnadhi et al. [14] demonstrate how linked data representations can be supported by ODPs with the central operations of *pattern flattening* and *view expansion* (in Sec. 2.2 we call these operations *lowering* and *lifting*) implemented using SPARQL UPDATE queries to transform compact linked data formats (views) to pattern representations, and vice versa. They also show how graph pattern conformance can be checked using SHACL [13].

This resembles the framework described in Sec. 2.2. However, a benefit of our approach is that, while all queries and format descriptions in the referenced work appears to be hand-crafted, similar lifting and lowering format descriptions and transformation can be automatically generated given an appropriate template specification. To demonstrate the abilities of OTTR templates for linked data publication, we have built the user-facing iChessGameReport template with which individual chess games may be expressed. (This template, and its nested templates, are equal in form to the Chess Game pattern templates, but are designed to take individuals and data values as input, rather then classes and properties.) The template may be found at http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/iChessGameReport. From this page different lifting and lowering queries and formats are available, as described in Sec. 2.3. Using the template instance format we can now compactly represent chess game instances which may be expanded to a full OWL pattern. Fig. 3d contains an instance of

[10] See http://ontologydesignpatterns.org/wiki/Submissions:ChessGame

the iChessGameReport template, available at `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/chess/iChessGameReportExample`. Note that template instances can be regarded as "standardised" pattern views, representing patterns in a compact format using a specific vocabulary. We do recognise that this format may not be fit for linked data publication and that user-defined template views are necessary; see also the future work section in Sec. 5.

Finally, we note that OTTR templates can also be used to identify pattern instantiations or template instances. For instance, using the generated SPARQL query from the ScopedDomainRange we can successfully extract all the macro applications from the published version of the Chess Game ODP [14].

4 Discussion and Related Work

We now highlight the main benefits and shortcomings that are inherent to the template mechanism and the representation language of OTTRs, and compare them with related work. As for benefits:

- OTTRs provide a simple, but powerful abstraction mechanism based on the well-known concept of nested non-cyclic macros and syntactic substitutions. This allows complex ontology expressions to be compactly represented by a naturally compositional structure which we believe supports more efficient construction and maintenance of ontologies following "don't repeat yourself" (DRY) principles.
- OTTR templates lets patterns to be explicitly identified as such and clearly encapsulated. This improves provenience and interoperability between ontologies using the same or related templates, as patterns need not be discovered.
- The implicit mapping between the template head and the body provides the basics for an extensible framework for handling semantic data that can represent and transform data on and between different formats and abstractions.
- Templates are formally defined as parameterised ontologies. This allows the semantics of the pattern to be verified using regular ontology reasoners. Furthermore, it makes the organisation of templates and the study of relations between them essentially an extension of the same well-studied issues regarding ontologies, and familiar terminology and theoretical machinery can be reused.
- OTTRs can be compactly represented in RDF as OWL ontologies using the OTTR vocabulary. This allows us to leverage the stack of existing W3C languages and tools, such as ontology editors and reasoners.
- As the expansion mechanism is based on syntactic substitutions, templates can take any RDF resource as input, i.e., classes, properties, individuals and data values, including also resources from the "logical" OWL and RDF(S) vocabularies.

As for shortcomings, some of the benefits have a negative dual side:

- The simple nature of our macro mechanism leaves OTTRs with limited expressivity: for instance, they do not allow for loop structures or conditionals and they do not return values. As an example, we cannot currently apply a template to all the subclasses of a given class without explicitly listing all the subclasses. Current and future work is directed at allowing for more complex and efficient expressions, and at precisely delimiting the expressive power of OTTRs.

- The compact representation of OTTR templates as RDF graphs using the notion of graph neighbourhood, results in a somewhat implicitly defined head and body of the template. This again requires that an RDF document can only contain a single template. It would be convenient to be able to collect multiple templates in one document, and to package templates which only are used by one ontology together with that ontology.
- In the RDF representation of OTTRs regular RDF resources play the role of variables. This means that special care must be taken when minting parameter variables, since upon instantiating the template *all* occurrences of the parameter variable will be replaced with the argument value. Elements from established vocabularies, such as the OWL vocabulary [19], should be avoided as variables or used with extreme care. Less obvious potential problem variable values are literals, where the same value may unintentionally be used in different contexts, e.g., as cardinality restrictions on properties.

A predecessor and inspiration to the current form of templates dates back to 2008 [12]. A recent paper by authors of the current paper presents a formal definition of templates and investigates their formal properties: using templates as macros, queries, and for data exchange; and reasoning over templates [6]. The paper at hand is the first account of the practical aspects of OTTR templates.

The Ontology Pre-Processor Language (OPPL) [11] is similar in function, but different in form to OTTRs. Like OTTRs, OPPL patterns are parameterised ontology expressions which can be nested and can specify pattern instances and patterns directly in OWL ontologies. OPPL is a more powerful language than the template mechanism allowing OPPL patterns to return values, which supports a more elegant composition of patterns. On the other side, as OPPL was originally designed as an ontology manipulation language for adding and removing ontology axioms, OPPL patterns are expressed as a series of OWL axiom insertions. This, and the fact that the OPPL pattern is represented "in verbatim" as an OPPL script in OWL annotation properties, places the pattern out of reach for ontology reasoners and requires the correctness of the pattern to be checked by reasoning on the effects of applying the pattern, rather that the pattern itself. Also, it is not clear if formal semantics for OPPL patterns are developed. The application focus of OPPL is somewhat different from OTTRs: OPPL patterns are intended to be fully expanded once they are used in the ontology. In contrast, we believe that OTTR template instances can appear in ontologies as instances lifted or lowered to the abstraction level suited for the given user. For instance, an ontology expert may prefer to examine an ontology formatted as a set of OWL axiom OTTR templates, while the domain experts might prefer to see only the user-facing template instances. Additionally, OPPL patterns are limited to OWL expressions in Manchester syntax [10], while OTTRs supports RDF macros and is designed to be applicable in a larger framework, cf. Sec. 2.2.

XDP [7], built on top of WebProtégé, provides a convenient graphical tool for selecting and instantiating templates using a template-based or specialisation-based approach, but does not offer additional capabilities for ODP instantiation at scale.

The M^2 mapping language [18] extends the OWL Manchester syntax [10] with ontology pattern descriptions to include direct references into spreadsheets for translating spreadsheet data into ontologies. It has hence a more narrow focus than OTTRs, but makes direct use of the underlying representation language in a similar manner as OTTRs.

Taking a broader approach, Tawny-OWL [16] provides an environment for building OWL ontologies using Clojure, with all the advantages of using a fully fledged programming language.

5 Conclusion and Future Work

We have presented the simple, but novel technique of using RDF(S) and OWL for representing *Reasonable Ontologies Templates (OTTRs)*. OTTRs are ontology macros or templates that provide an extensible and transparent framework for creating and using ontology design patterns in the design and construction of ontologies. Templates are in essence n-ary relations that relate a simple tabular input format, defined by its template head, to a rich ontological structure in the template body, possibly via compositions of other templates. From the template head, different tabular data input formats may be generated. This allows ontology experts to design "user-facing" templates for the purpose of collecting domain knowledge from expert users on tabular format. The ontological definition of the template is produced by expanding the template to a regular ontology. Bulk transformation specifications of the input data to ontology format may also be generated from the template. Templates are formulated in OWL using a special purpose OTTR OWL vocabulary. By virtue of being OWL ontologies, templates may be shared, reused, and debugged using existing semantic web technologies and tools. Additionally, the OTTR vocabulary supports simple type checking and terse formats for template specifications. A prototype implementation for expanding and generating various specifications from templates is available online at http://www.ottr.xyz.

Future work. The present proposal for templates has been developed in close interaction with industrial user communities, and we intend to apply it to various existing enterprise ontologies in the immediate future. This will serve to evaluate, verify and refine the concept, and will help us develop an efficient and reliable set of tools and web services. We believe that templates can be important for development and use of open, validated modelling patterns, as is required for shared models, and for enabling ontology-based collaboration. In order to develop templates that cover typical needs of industrial users, we will work with standardisation bodies and make these templates available through a public repository. This should lower the cost of translating existing data into ontology, opening up the benefits of ontology-based methods to new users.

To support this work, tools and methods for constructing, structuring and managing templates are necessary. To this end, we intend to further develop the prototype implementation to support more input representation and validation formats, such as spreadsheets and RDF graph shape validations, and to develop a Protégé plugin for developing and applying OTTR templates to ontology development.

We also intend to continue the initial efforts on the logical properties of templates as is found in [6]. A template can dually be regarded as a macro or as a (higher-order) query; whether one is asking for a pattern to be added to an ontology or asking for occurrences of the pattern in the ontology is a difference of use of the template, and not of the template itself. Furthermore, this duality makes it easy to extend the use of templates to adding a pattern conditionally on another pattern occurring in the ontology, or to use templates as constraints on ontologies. The latter observation can for instance be used to implement

pattern views, allowing template instances to be specified using a custom vocabulary. A different problem is if this possible to (elegantly) represent in OWL.

References

1. E. Blomqvist, K. Hammar, and V. Presutti. *Engineering Ontologies with Patterns – The eXtreme Design Methodology*, chapter 2, pages 23–50. Volume 025 of Hitzler et al. [9], 2016.
2. D. Brickley and R. Guha. RDF Schema 1.1. Technical report, W3C, 2014.
3. R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Technical report, W3C, 2014.
4. N. Drummond et al. Putting OWL in Order: Patterns for Sequences in OWL. In *OWLED*, 2006.
5. J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. Technical report, W3C, 2007.
6. H. Forssell, D. P. Lupp, M. G. Skjæveland, and E. Thorstensen. Reasonable Macros for Ontology Construction and Maintenance. In *DL Workshop*, 2017.
7. K. Hammar. Ontology Design Patterns in WebProtege. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, 2015.
8. K. Hammar and V. Presutti. Template-Based Content ODP Instantiation. Workshop on Ontology and Semantic Web Patterns, WOP 2016.
9. P. Hitzler et al., editors. *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, volume 025. IOS Press, Amsterdam, 2016.
10. M. Horridge and P. F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax. Technical report, W3C, 2012.
11. L. Iannone, A. L. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009.
12. J. W. Klüwer, M. G. Skjæveland, and M. Valen-Sendstad. ISO 15926 templates and the Semantic Web. W3C Workshop on Semantic Web in Oil & Gas Industry, 2008.
13. H. Knublauch and D. Kontokostas. Shapes Constraint Language (SHACL). Technical report, W3C, 2017.
14. A. Krisnadhi et al. *Ontology Design Patterns for Linked Data Publishing*, chapter 10, pages 201–232. Volume 025 of Hitzler et al. [9], 2016.
15. A. Krisnadhi and P. Hitzler. *Modeling With Ontology Design Patterns: Chess Games As a Worked Example*, chapter 1, pages 3–21. Volume 025 of Hitzler et al. [9], 2016.
16. P. Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. In *OWLED*, 2013.
17. B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. Technical report, W3C, 2012.
18. M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. M2: A Language for Mapping Spreadsheets to OWL. In *OWLED*, 2010.
19. P. F. Patel-Schneider and B. Motik. OWL 2 Web Ontology Language Mapping to RDF Graphs. Technical report, W3C, 2012.
20. V. Presutti and A. Gangemi. Content Ontology Design Patterns As Practical Building Blocks for Web Ontologies. In *ER*, pages 128–141. Springer, 2008.
21. D. Vrandečić. Explicit knowledge engineering patterns with macros. In *Proceedings of the Ontology Patterns for the Semantic Web Workshop at the ISWC 2005*, 2005.
22. Wikipedia. Software design pattern—Wikipedia, the free encyclopedia, 2017. [Online; accessed 27-July-2017].